

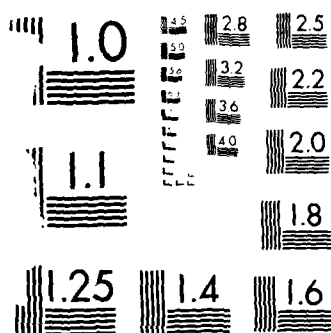
AD-A165 322 MONITORING AN ADA SOFTWARE DEVELOPMENT(U) MARYLAND UNIV 1/1  
COLLEGE PARK DEPT OF COMPUTER SCIENCE  
V R BASILI ET AL. APR 84 N00014-82-K-0024

UNCLASSIFIED

F/G 9/2

ML





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

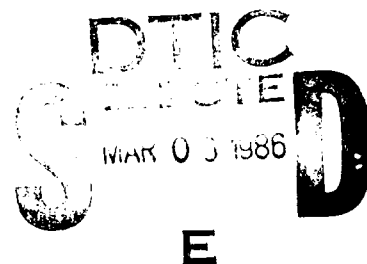
①

AD-A165 322

## MONITORING AN ADA SOFTWARE DEVELOPMENT

Victor R. Basili  
Shih Chang  
John Gannon  
Elizabeth Katz  
N. Monina Panlilio-Yap  
Connie Loggia Ramsey  
Marvin Zelkowitz

John Bailey  
Elizabeth Kruesi  
Sylvia Sheppard



University of Maryland      General Electric Company\*

Ada evolved from a desire within the Department of Defense to have a standard language for the development of real-time and large scale systems. In addition to providing features needed by those types of systems, Ada supports structured programming, data abstraction, modularity, and information hiding. Research with these techniques indicates that their use should improve the quality of the software development process and its product. While, programmers who are most familiar with various assembly languages and FORTRAN may use structured programming, generally they are not familiar with the other concepts. The problems with training programmers in Ada and its associated design and programming methods and then redeveloping current systems in Ada is unknown.

In order to understand the effect of using Ada, the University of Maryland and the General Electric Company began a joint project. The purpose of the project is to monitor the use of Ada in an industrial software development project. In particular, we identify areas of success and difficulty in learning and using Ada as both a design and coding language. Our results indicate where emphasis should be placed in Ada training and in the development of tools and techniques for use with Ada. We also identify metrics used to evaluate and predict the cost, quality, and maintainability of Ada programs.

Copies of the newsletters may be obtained from Dr. Victor R. Basili, Department of Computer Science, University of Maryland, College Park, MD, 20742. Feedback regarding our approach, goals, and results is welcome.

\* John Bailey and Elizabeth Kruesi are now with Software Metrics, Inc. Sylvia Sheppard is with Booz, Allen, and Hamilton, Inc.

This research is supported by the Office of Naval Research (ONR) under ONR contract #N00014-92-K-0024 to the University of Maryland with funding from ONR and the Ada Joint Program Office. Ada is a trademark of the Department of Defense, Ada Joint Program Office.

This is the third newsletter concerning this project. April 1984.

This is for distribution

## Background

### *Case Study Organization*

This case study is driven by a set of goals that were developed to answer questions about the Ada language and its use. Setting out with a set of goals, a framework is developed for establishing why each data item is collected and how it is used to evaluate the project being studied. This approach minimizes the chance that needed data will not be collected and permits the interpretation of the data relative to the goals set. The goals also provide an organization for the data collection process. The approach consists of six steps:

- 1) the development and categorization of a set of goals,
- 2) the development of a set of questions of interest or hypotheses based upon those goals that attempt to quantify the abstractions of the goals,
- 3) the development of metrics and data distributions that answer the questions,
- 4) the development of forms and other mechanisms for collecting the data,
- 5) the actual data collection process, and
- 6) the validation and analysis of the data.

[Basili, Weiss 82] contains a detailed discussion of this approach to data collection.

The primary goal of this endeavor, to study an Ada project in order to make recommendations, was broken down into more specific goals to guide the study. These goals were divided into four major categories: changes and effort goals, Ada and PDL goals, data collection goals, and metric goals. Each goal within a category was associated with a series of questions whose answers might help meet that goal. These questions did not include every question one might ask, but they were meant to be representative of the questions of greatest interest. Each question had a list of data sources, such as forms, static analyzers, or human evaluations, to guide the data collection process. The complete list of goals, without the questions, is given in the appendix.

The source of much of the effort and change data was a set of forms that were developed to gather information about the software development process. Most of these forms were adapted from the NASA Software Engineering Laboratory [SEL 82]. They were completed by the programming team and validated by the monitoring team before their entry into an automated database. The preliminary analysis of that data is given in the subsection "Effort and Change." In addition, several subjective evaluations were made by various people at various stages of the development. That data has not yet been analyzed. Finally, a parser, which checks syntax of both the design and code and takes rudimentary measurements, has processed the final product. The analysis of that data is in the subsection "Ada Use." The goals and questions directed how this analysis was done.

### *Project Development*

The project studied involved the redesign and reimplementation of a portion of a satellite ground control system originally written in FORTRAN. Four programmers were chosen for their diverse backgrounds: the lead programmer/manager knew the application, FORTRAN, and assembler; the senior programmer also knew other languages such as COBOL, PL/I, Lisp, ALGOL, and SNOBOL; the junior programmer

had just earned a B.S. in computer science and was was fluent in Pascal and other block-structured languages; the librarian had only brief exposure to FORTRAN. They were given a month of training in Ada and the programming practices they were expected to use: design and code walkthroughs and structured programming. They practiced using the NYU Ada/Ed interpreter on sample programs and began designing the system. The design was written using an Ada-like PDL which specifies Ada packages and subprograms and their interfaces as well as abstract statements for program functions. Although the PDL is designed to be processable, a processor was not available at design time. The design evolved into Ada code which was processed by the Ada/Ed interpreter; however, the entire project could not be interpreted as a unit due to size constraints with the interpreter. The design and coding phases of the project extended from March 1982 to January 1983. Some testing of the system was done during the summer of 1983 using the ROLM compiler; however, the entire system has not been tested. In addition, since there was no test plan developed before or during the project, we cannot evaluate the testing process.

## Data Analysis

### *Effort and Change*

Preliminary analysis has been done on the effort, change and fault data. Given the overall expenditure on the project, relative to most projects, a large amount of time was spent on training, about 20%. Also, the effort spent on each of requirements and design was greater than the effort spent on coding. Little time was spent on testing. However, it must be stressed that the project's development cycle was not completed. As it became apparent that a full-fledged compiler would not be available for use on this project, the programmers' enthusiasm waned. Some low-level procedures were not written, and very little of the system was more than unit tested. Therefore, the percentage of time spent on various phases may be misleading.

While the programmers were given a more extensive training program than might be considered normal, it should be noted that Ada training costs will most likely be higher than average and must be considered when planning early developments using Ada. In addition, many of the concepts incorporated into Ada were not used by the programmers. For example, even though data abstractions and their use were taught during the training program, they were not used in the early work and were used in later coding only after an understanding of the project design and the concepts of data abstraction were reinforced. Training must be oriented toward the concepts behind Ada and how they are supported by the language rather than toward the language with reference to the concepts. A related study concerning training in Ada is described in the section "Training Study."

After the training, the project programmers began to design and then code. Changes were documented from the time that each piece of design was reviewed. The change report forms show that most changes were design (32%) and code (61%) changes and that there were few requirements changes (7%). In addition, most of the changes were fault corrections (57%) and improvements for clarity, maintainability, and readability (23%). The need for change was determined in less than an hour for almost all of

the changes, and the time to design and implement the change was one hour or less for almost all of the changes. Since the code was not thoroughly tested, we do not know whether an even higher percentage of fault correction type changes may be needed.

Analysis of the fault report forms indicates that 72% of the faults entered the system during the coding stage and 24% during the design stage. However, since the design was not machine checked and the programmers did not go back to the design to determine whether a fault discovered in the code was present in the design, preliminary visual analysis indicates these percentages are probably closer to 50/50. Most of the faults were incorrect code (79%) and incorrect design (16%). The majority of the forms indicated that the use of Ada contributed to the fault and most of these were syntactic faults. Programmers claimed that the Ada language reference manual or class notes explained the features clearly in most cases and that they understood the features but did not apply them correctly. To correct the fault, programmers usually remembered how the features should be applied or obtained information from another programmer. Most faults took less than fifteen minutes to isolate and as little time to correct. The activities used to detect and isolate faults were mainly compilation, design reading, design walkthrough, code reading, or some combination of these.

The above information seems to indicate that most of the faults discovered were trivial. Without having done thorough testing, it is impossible to say how many more serious and change-resistant faults may still exist in the code.

The faults were also classified as language, problem and clerical. Language faults were those which involved the syntax or semantics of a feature or those which involved the concept behind a feature. The problem category involved those faults due to a lack of understanding of the approach or solution domain but not related to the language. Clerical faults included those due to carelessness, e.g. typographical faults. Eighty-six percent of the faults were language faults, and furthermore, 89% of these were merely syntax faults. This explains why so many of the faults took so little time to correct. Twenty-seven percent of the language faults were semantic faults. Most of the faults involving requirements were problem faults, and most of the faults involving incorrect design or code were language-related faults.

Several Ada language features were involved in faults. Most common among these were low-level syntax (e.g. semicolon, parenthesis, assignment) and loops. There were also a considerable number of faults involving tasks, separate compilation, generics, procedures and functions, parameters, and declarations. A smaller number of faults were related to exceptions, types, packages and several other features. There were only a few concept faults, and these involved tasking, exceptions and packages. Parameters, generics and compilation units together accounted for 53% of the semantic faults. These results suggest that further training in the concepts of Ada, along with a language-based editor, might eliminate many of the type of faults found in this project.

#### *Ada Use*

All of the design and code has been processed. There are 11145 lines of Ada source (including comments) and 7406 lines of PDL source, some of which evolved into Ada source. The Ada code consists of 2913 statements (1064 declarations and 1849

executable statements). There are 50 program units (packages, tasks, or subprograms), 18 of which are packages.

Early design reviews showed that the design was functional rather than object-oriented. This subjective opinion is supported by an analysis of the packages. Of the eighteen packages, two were common blocks of definitions, three were libraries of functions, eleven were encapsulated data types with private types and operations, and the remaining two had defined types but made the representation of the type visible. Nine of the packages defining encapsulated types were device drivers, one encapsulated mathematical functions for different types of data, and the remaining package definition had no body. Device drivers and math libraries are used in existing software systems. No new fully encapsulated types were declared. Therefore, the programmers did not seem to find new abstract data types [Gannon, et al. 83].

One reason for use of a functional design might be that the requirements are detailed and functionally oriented. It was probably easier for the programmers to design the system functionally based on those requirements than to abstract back from the requirements to a level where they could see other design alternatives [Duncan, et al. 84]. In addition, since the programmers had more experience with FORTRAN than any other language, they may have been constrained by their previous language experience [Booch 81]. Training for alternative design approaches and other software engineering concepts supported by Ada must come early in development. This training probably should precede training in the Ada language, since it impacts the early design decisions and perhaps the requirements analysis phase.

Two of the goals of the project (II.2 and II.5) relate to the use of the Ada language. As a first step, we have examined each programmer's use of executable statements. Of the Ada executable statements (1849), 16% (301) were written by the lead programmer/manager, 43% (795) by the senior programmer, 36% (671) by the junior programmer, and 5% (82) by the librarian. Any comparison of language use will probably not include the librarian because he wrote relatively few executable statements. In discussing each programmer's use of Ada, we indicate which percentage of each programmer's executable statements is involved in order to normalize the data.

The librarian was the only team member to use a discernibly limited subset of Ada executable statements. He used assignments (49%), ifs (20%), returns (16%), loops (13%), and raised exceptions (2%). This use seems to be appropriate for the subprogram he wrote. The other programmers used almost every type of executable statement. The code statement was probably not appropriate for this application, and they avoided the goto statement as well. However, the lead and senior programmers used 10 and 20 (3.3% and 2.5%) exit statements respectively. The exit can be considered a restricted goto. Only the senior programmer used the abort statement, and the lead programmer used 14 (4.7%) pragmas while the junior programmer used 2 (0.3%).

Little distinction between programmers can be made using this data at this level of analysis. We are investigating more detailed measures of language and data use. We also will try to develop further measures of the use of Ada concepts such as exception handling, tasking, and abstraction [Basili, Katz 83]. As our analyzer becomes more sophisticated, we hope to further characterize the use of Ada on this and other projects.

## Training Study

In a related empirical study, we compared two approaches to teaching the Ada language. The goal was to discover an effective way to teach students the use of Ada as a vehicle for applying information hiding and data abstraction to software development. The fifty-four participants in the study were enrolled in an advanced undergraduate Ada class at the University of Maryland. Baseline data was gathered on every student, including programming aptitude scores. The class was then randomly divided into two sections. One section was taught the language features first, approximately in the order that they are presented in the language manual. They were then shown how packages can be used to encapsulate objects, resources, and types when a system is first designed. The other section was taught these principles of encapsulation first and used the Ada package to express designs before the lower-level language features were presented. Eventually, the same set of lectures was presented to both sections.

We initially hypothesized that the section which learned design first would produce more modifiable programs. However, the lack of complete, executable examples during the entire first half of the course appeared to hamper a complete understanding of the concepts. Ultimately, the high variability among the students masked any large differences between the sections. However, some interesting differences in the correlations between background data and the success of the students in each section were revealed. This experience suggests that the optimal approach would probably involve tailoring a curriculum to each student's background and experience. However, a combination of the two approaches, where complete examples are presented with emphasis on design considerations, might be appropriate even when teaching professional programmers.

## Conclusions

The Ada language is a medium for supporting certain design concepts. It is important that those concepts be taught and motivated, possibly even before the language is taught. Training should be tailored to the past experience of the programmers. The programmers on this project had trouble with data abstraction and information hiding and distinguishing between detail and precision particularly when designing. Only after the project was complete did they understand the importance of methodology and how it should be used. Their overall design was more like than unlike a FORTRAN system design. However, the requirements were already functional.

In this project, the programmers used most of the language features but not necessarily as they were intended by the language designers. There were a large number of language errors made, and these errors were syntactic, semantic, and conceptual. Most of the errors involved the more Ada-specific features. Due to the learning curve, we were unable to judge the impact of Ada on costs, schedules, or milestones. However, it is clear that many support tools are needed. These tools include a structured editor, data dictionaries, call structure and compilation dependency tools, cross references, and other means of obtaining multiple views of the system. In addition, a PDL processor with interface checks, definition and use relation lists, and metrics would be helpful in the early stages of development.

## Further Research

Some further analysis will be done with this data. The design and code will be studied to determine whether previous experience influences a programmer's use of Ada. Since the project was not finished and the product not tested fully, we expect few concrete results from this data. However, we plan to use this data to aid in the evaluation of analysis tools we will develop. After building these tools, we plan to look at other projects developed in Ada for further indications of the effect of Ada. Recommendations will then be made on which metrics are most useful, what aspects of training must be stressed, and what influence the use of Ada might have on the software development process. We encourage comments on all aspects of this project and will continue to publish newsletters or papers concerning our results.

## References

[Basili et al. 82]

Victor R. Basili, John D. Gannon, Elizabeth E. Katz, Marvin V. Zelkowitz, John W. Bailey, Elizabeth E. Kruesi, and Sylvia B. Sheppard, "Monitoring an Ada Software Development Project," *Ada Letters* II, 1 (July 1982), 1.58-1.61.

[Basili, Katz 83]

Victor R. Basili and Elizabeth E. Katz, "Metrics of Interest in an Ada Development," *IEEE Workshop on Software Engineering Technology Transfer*, Miami, FL, April 1983, pp. 22-29.

[Basili, Weiss 82]

Victor R. Basili and David M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *Computer Science*, Univ. of Maryland, 1982, UOM-1235.

[Booch 81]

Grady Booch, "Describing Software Design in Ada," *SIGPLAN Notices*, Vol. 16, No. 9, Sept. 1981, pp. 42-47.

[Duncan, et al. 84]

A. G. Duncan, J. S. Hutchison, J. B. Bailey, T. M. Chapman, A. Fregly, E. E. Kruesi, T. McDonald, S. B. Sheppard, "Communications System Design Using Ada," *Proc. 7th Intl. Conf. on Software Engineering*, Orlando, FL, March 1984, pp. 398-407.

[Gannon, et al. 83]

John D. Gannon, Elizabeth E. Katz, and Victor R. Basili, "Characterizing Ada Programs: Packages," *The Measurement of Computer Software Performance*, Los Alamos National Laboratory, August 1983.

[SEL 82]

Software Engineering Laboratory, SEL-81-104, *The Software Engineering Laboratory*, NASA Goddard Space Flight Center, February 1982.

## Appendix

The purpose of these goals is to direct the study of this Ada project. Complete copies of the list of goals and questions may be obtained from the authors.

### I. Changes and Resources

- I.1: Characterize the effort in the project.
- I.2: Characterize the changes.
- I.3: Characterize the faults.
- I.4: Characterize Ada faults.
- I.5: Characterize the other faults.
- I.6: Characterize the non-error changes.

### II. Ada and PDL/Ada

- II.1: Evaluate the effect of using an Ada-like PDL with respect to the goals of a PDL.
- II.2: Determine which subsets of Ada features are used naturally.
- II.3: Determine the effect of using an Ada-like PDL when Ada is the language of implementation.
- II.4: Determine how Ada works for this application.
- II.5: Characterize the programmers and associate their backgrounds with their use of Ada.
- II.6: Determine whether there are aspects of Ada that contribute positively to the design and programming environment.

### III. Data Collection

- III.1: Evaluate the data collection and validation process.

### IV. Metrics

- IV.1: Select a set of static metrics for the APSE.
- IV.2: Develop a set of size metrics for the APSE.
- IV.3: Develop a set of control metrics for the APSE.
- IV.4: Develop a set of data metrics for the APSE.
- IV.5: Select a set of dynamic metrics for the APSE.
- IV.6: Develop a set of test coverage metrics for the APSE.
- IV.7: Develop a set of execution metrics for the APSE.
- IV.8: Select a set of software development process metrics for the APSE.
- IV.9: Determine the effectiveness of the predictive power of certain measures during development.
- IV.10: Develop a subjective evaluation system for evaluation of program and design characteristics that are not practically or easily measured in other ways.
- IV.11: Provide a data base for future Ada projects to be used to predict some properties of those projects.

Dist  
A-1

DTIC

FILMED

4-86

END